
Salamander: Dynamic Network Reconfiguration

David Montana and Talib Hussain
BBN Technologies



Objective

- Dynamically reconfigure a network so as to continually provide optimal quality of service (QoS)
- Network maintains the required QoS for critical users and applications despite
 - ◆ attacks on nodes and links
 - ◆ failures of nodes and links
 - ◆ changes in loads and traffic patterns
 - ◆ changes in requirements and priorities



Approach

- Define an evaluation function that associates a score with any potential network configuration
 - ◆ priority-weighted aggregate measure of how well QoS requirements of different users/applications met
- Compute the evaluation function using the *ns* network simulator to determine application-level QoS
- Use a genetic algorithm to search the space of possible network configurations for the optimal



Network Data: The Givens

- Nodes
 - ◆ number of send and receive modems
- Fixed Links
 - ◆ from node and to node
 - ◆ bandwidth and inherent transmission delay
- Satellites (Reconfigurable Links)
 - ◆ number of channels and bandwidth per channel
- Offered Loads
 - ◆ from node and to node
 - ◆ amount of data transmitted and statistics on distribution
 - ◆ TCP or UDP
 - ◆ priority
 - ◆ required QoS measures (latency and dropped packets)



Evaluation Function

$$\sum_{i=1}^n \frac{1}{\rho_i} \left(\alpha (\lambda_i^r - \lambda_i^s) + \beta (\delta_i^r - \delta_i^s) \right)$$

where:

- n is the number of loads
- λ_i^r is the required maximum latency for load i
- λ_i^s is the actual latency achieved during simulation for load i
- δ_i^r is the required minimum packet drop rate for load i
- δ_i^s is the actual drop rate achieved during simulation for load i
- α and β are fixed values used to determine the relative importance of latency versus packet drop rate



Experiments

- Three classes of networks with known optimal solutions
 - ◆ ClientServer: 5 nodes, 4 clients and a server, with different variations of loads and disabled resources
 - ◆ Bottleneck: n nodes transmitting via UDP to m other nodes, squeezing through a single overtaxed link
 - ◆ Ring: theoretically interesting but unrealistic, worst-case scenario for genetic algorithm
- Execute genetic algorithm for fixed number of evaluations and determine quality of solution
- Run on a single 850MHz laptop
- Results based on single run, not yet scientific



Timing Results

Problem	Nodes/Links	Search Space	Evaluations	Time (H:M)
ClientServer1	5 / 4	4.1×10^4	< 500	< 0:04
Bottleneck1	12 / 5	3.3×10^8	< 500	< 0:07
Bottleneck2	17 / 5	1.2×10^{10}	< 500	< 0:29
Bottleneck3	22 / 5	1.8×10^{11}	< 1000	< 0:43
Bottleneck4	22 / 6	1.4×10^{13}	< 2500	< 1:45
Ring8	8 / 4	4.1×10^4	< 500	< 0:06
Ring12	12 / 6	7.3×10^9	< 10,000	< 2:31
Ring16	16 / 8	2.7×10^{14}	< 50,000	< 17:38
Ring20	20 / 10	1.7×10^{19}	> 100,000	> 44:24



Timing Results Caveats

- Will be much faster when we
 - ◆ optimize the code
 - ◆ optimize the algorithm
 - ◆ use distributed optimization
 - ◆ settle for good solution rather than best solution



Future Directions

- Current contract
 - ◆ Optimization speedups
 - remove *ns* startup costs
 - continue with distributed optimization
 - heuristic initialization of population
 - ◆ Documentation of results
- Potential follow-on work
 - ◆ Integration with real network, using statistics from network as inputs and implementing results in network
 - ◆ Massively distributed computing
 - ◆ Other types of reconfigurable components (besides satellites channels)

